LPMARL: Linear Programming-based Task Assignment for Hierarchical Multi-agent Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Training Multi-Agent Reinforcement Learning (MARL) with sparse rewards is challenging due to complex agent interactions. We propose Linear Programming-based hierarchical MARL (LPMARL), which integrates constrained optimization into MARL for effective coordination. LPMARL operates in two stages: (1) solving agent–task assignment via a Linear Program with state-dependent costs from a Graph Neural Network (GNN), and (2) solving cooperative sub-games among assigned agents. Both the LP generator and low-level policies are trained end-to-end by differentiating through the optimization layer. Experiments show that LPMARL achieves effective task allocation and sub-policy learning across diverse cooperative games.

1 Introduction

Multi-Agent Reinforcement Learning (MARL) is promising for controlling complex distributed systems, but training under sparse rewards remains challenging because outcomes depend on long-term interactions among agents. Effective methods must capture the relation between sequential actions and delayed rewards to enable efficient decision-making.

A widely studied direction is hierarchical MARL, which decomposes complex problems into subtasks through high-level and low-level policies. Some approaches rely on pre-defined high-level action spaces under the semi-MDP framework [Sutton et al., 1999], such as using temporally abstracted task-selection policies [Tang et al., 2018], centralized allocation strategies [Ahilan and Dayan, 2019], or exploration-based high-level policies to constrain the low-level action space [Liu et al., 2021]. While these methods promote the formation of task-dependent sub-groups, their low-level policies are often trained individually, limiting cooperation among agents.

Other works attempt to remove the need for explicit high-level action spaces by learning latent *roles* or representations. For example, role-based methods [Wang et al., 2020a, Yang et al., 2022], action representation learning [Wang et al., 2020b], and intrinsic reward shaping [Yang et al., 2019] provide flexible abstractions for coordination. However, these latent strategies can be difficult to interpret, require careful selection of the number of roles, and often transfer poorly across environments.

A complementary line of research formulates agent–task allocation as a constrained optimization problem with state-dependent parameters [Carion et al., 2019]. Such formulations offer structure but typically rely on rule-based low-level policies, which limits cooperation.

Motivated by these limitations, we propose **LPMARL**, a hierarchical framework with two stages: (1) solving agent–task assignment via Linear Programming (LP) with state-dependent costs from a Graph Neural Network (GNN), and (2) solving local cooperative sub-games among assigned agents.

NeurIPS 2025 Differentiable Learning of Combinatorial Algorithms Workshop (NeurIPS DiffCoALG 2025).

Both the LP generator and low-level policies are trained end-to-end through a differentiable optimization layer, enabling interpretability, generalization, and practical decentralized execution.

The contributions of this paper are as follows:

- **Interpretability.** LP acts as an algorithmic prior, embedding explicit objectives and constraints into the high-level policy.
- **Training.** A differentiable pipeline jointly optimizes representation learning, LP solving, and cooperative control.
- **Practicality.** GNN parameterization supports generalization to varying agents, tasks, and constraints, and a decentralized variant amortizes LP assignment for scalable execution.

2 Preliminaries

2.1 Problem formulation

LPMARL is designed to address cooperative MARL problems consisting of multiple tasks and joint constraints. Here, **tasks** denotes specific sub-goals that agents must accomplish for the overall success of the game. For example, in the cooperative navigation environment [Liu et al., 2021], agents must select distinct landmarks (task allocation) and reach them without collision (task completion).

We model this setting with a hierarchical Dec-POMDP [Tang et al., 2018], defined as $< \mathcal{N}, \boldsymbol{\mathcal{S}}, \{\mathcal{O}_i\}_{i=1}^N, \{\mathcal{A}_i^h\}_{i=1}^N, \{\mathcal{A}_i^l\}_{i=1}^N, \mathcal{R}, \mathcal{T} >$, where $\mathcal{N}$ is the set of agents, $\boldsymbol{\mathcal{S}}$ the state space, $\{\mathcal{O}_i\}$ the observation space, $\{\mathcal{A}_i^h\}$ the high-level (task) action space, $\{\mathcal{A}_i^l\}$ the low-level (primitive) action space, $\mathcal{R}$ the reward function, and $\mathcal{T}$ the transition probability.

At each timestep, agent i receives a partial observation $o_i = o_i(\boldsymbol{s})$ from the global state $\boldsymbol{s} \in \boldsymbol{\mathcal{S}}$. It then selects a high-level action a_i^h through its policy $\pi_i^h : \mathcal{O}_i \to \mathcal{A}_i^h$. Conditioned on this high-level action, the agent chooses a low-level action a_i^l using $\pi_i^l : \mathcal{O}_i \times \mathcal{A}_i^h \to \mathcal{A}_i^l$, which is executed in the environment.

Agents receive intrinsic rewards $r_i^l(\boldsymbol{s}, \boldsymbol{a}^l; a_i^h)$ for completing sub-goals, and a high-level reward $r^h(\boldsymbol{s}, \boldsymbol{a}^l)$ when the global task is successfully achieved. The objective is to learn high- and low-level policies that maximize the expected cumulative reward: $\mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$.

2.2 Implicit deep learning

Implicit deep learning is a framework that incorporates implicit rules, such as ordinary differential equations [Chen et al., 2018], fixed-point iterations [Bai et al., 2019], and optimization [Amos and Kolter, 2017], into a feed-forward neural network. This framework utilizes differentiable optimization layers that take problem-specific parameters as input and find optimal solutions based on objective functions constructed with given parameters. The output of these layers, serving as the optimization inductive bias, is then passed to subsequent layers to perform various tasks. The utilization of implicit deep learning provides a foundation for LPMARL, enabling differentiation through the embedding layer, LP layer, and policy network, facilitating efficient learning and coordination among agents.

3 Methodology

At each step, LPMARL proceeds in three stages: (1) constructing an agent–task score matrix via a GNN, (2) solving an agent–task assignment LP, and (3) solving cooperative sub-games among agents assigned to the same task (Fig. 1).

3.1 Constructing agent–task score matrix

Given global state $\boldsymbol{s} = \{s_k : k \in \mathcal{N} \cup \mathcal{M}\}$, we build a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with agent nodes $\mathcal{N}$ and task nodes $\mathcal{M}$. Edges represent agent–agent and agent–task relations. A message-passing

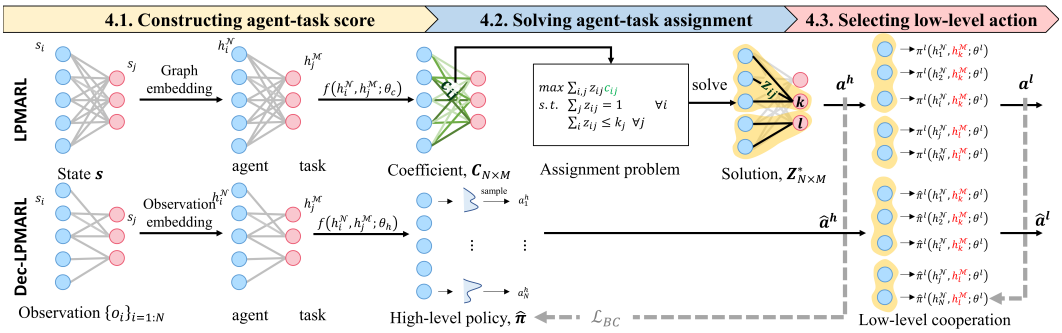

Figure 1: Overall decision-making framework of LPMARL

GNN [Battaglia et al., 2018] updates embeddings as

$$m_{ij}^{\mathcal{NN}} = f(h_i, h_j; \theta_g^{\mathcal{NN}}),\ ij \in \mathcal{E}_{\mathcal{NN}} \tag{1}$$

$$m_{ij}^{\mathcal{NM}} = f(h_i, h_j; \theta_g^{\mathcal{NM}}),\ ij \in \mathcal{E}_{\mathcal{NM}} \tag{2}$$

$$h_i' = f([\sum m_{ij}^{\mathcal{NM}} \| \sum m_{ij}^{\mathcal{NN}} \| h_i]; \theta_g^{\mathcal{V}}), \forall i. \tag{3}$$

Using updated embeddings $\{h_i\}$, we compute cost coefficients $c_{ij} = f(h_i^{\mathcal{N}}, h_j^{\mathcal{M}}; \theta_c)$,, where c_{ij} scores the allocation of agent i to task j. The resulting matrix C forms the objective of the assignment LP.

3.2 High-level policy: agent–task allocation

The centralized high-level policy solves the LP:

$$\text{maximize} \quad \sum_{i,j} c_{ij} \cdot z_{ij} \tag{4}$$

$$\text{s.t.} \quad \sum_{i} z_{ij} \le k_j,\ \forall j \in \mathcal{M} \tag{5}$$

$$\sum_{j} z_{ij} = 1,\ \forall i \in \mathcal{N} \tag{6}$$

$$0 \le z_{ij} \le 1. \tag{7}$$

The solution $Z^* = \{z_{ij}^*\}$ represents assignment probabilities, i.e., $\pi^h(a_i^h = j|\boldsymbol{s}) = z_{ij}^*$. Agents are then grouped by assigned task.

Decentralized policy. Centralized LP solving may be impractical at execution. We therefore introduce Dec-LPMARL, where each agent uses a GNN-based imitation policy $\hat{\pi}_i^h(a_i^h = j|o_i) = \hat{z}_{ij}^*$ trained to approximate Z^*. Details of this amortized policy are given in Appendix D.

3.3 Low-level policy

Once agents are grouped by tasks, each agent selects low-level actions conditioned on both its observation and assigned task:

$$\pi^l(a_i^l|o_i, a_i^h) = \text{softmax}\big[f(h_i^{\mathcal{N}}, h_{a_i^h}^{\mathcal{M}}; \theta_l)\big]_{a_i^l}. \tag{8}$$

Here θ_l are the parameters of the low-level policy, shared among agents within a group.

4 Training

We jointly train the parameters of the high-level policy θ_h and low-level policy θ_l by optimizing a weighted objective

$$\mathcal{J} = w \cdot \mathcal{J}_l(\theta_l) + (1 - w) \cdot \mathcal{J}_h(\theta_h),$$

where w is decayed from 0.9 to 0.1 during training to account for the higher sparsity of high-level rewards.

4.1 High-level policy

The high-level actor–critic is trained using sparse team rewards. The critic Q^h estimates long-term returns, while the policy gradient is computed by differentiating through the LP optimization layer. Since LP solutions can be piecewise constant, we follow Vlastelica et al. [2019] to approximate gradients using perturbed objectives (details in Appendix B).

4.2 Low-level policy

The low-level policy is trained to maximize the sum of sub-task rewards within each agent group. We adopt value function factorization Sunehag et al. [2017] to construct the critic Q^l, and optimize π^l via standard policy gradients (details in Appendix C).

5 Experiment

In our experimental evaluation, we address several key questions to assess the performance and capabilities of our approach. (1) Can LPMARL learn cooperative agent-task allocation? (2) Is LPMARL transferable to different problem sizes? and (3) Can Dec-LPMARL amortize the centralized task allocation optimization procedure using a decentralized imitation policy?

The primary objective of this study is to demonstrate how incorporating structural assumptions enhances cooperation. To achieve this, we compare our approach with non-hierarchical MARL algorithms, such as Qmix [Rashid et al., 2018], MADDPG [Lowe et al., 2017], and MAAC [Iqbal and Sha, 2019], which do not explicitly incorporate task allocation. Additionally, we evaluate the effectiveness of our proposed hierarchical decision-making scheme by comparing it with existing hierarchical MARL approaches, such as HSD [Yang et al., 2019], RODE [Wang et al., 2020b], and LDSA [Yang et al., 2022]. Through these comparisons, we aim to highlight the advantages and improvements offered by our approach in addressing cooperative problem-solving scenarios.

In addition, we consider two ablations of LPMARL: **Dec-LPMARL** and **No-LP**. Dec-LPMARL explores the feasibility of decentralizing LPMARL by decentralizing the task allocation process, allowing agents to make decisions based on local observations. No-LP serves as a comparison to evaluate the effectiveness of using LP as the high-level policy, where the global state is directly used as input for generating the high-level policy.

5.1 Constrained cooperative navigation

In order to investigate the high-level assignment of LPMARL, we produced a modified version of the cooperative navigation [Lowe et al., 2017], namely the constrained cooperative navigation. In this environment, N agents aim to occupy M ($M < N$) landmarks, where each landmark has its own capacity limit to accommodate the agents. In this environment, the agents receive a success reward only in the case when the agents occupy the landmarks while satisfying the capacity constraint. Additional experimental details are described on Appendix E.2.

Table 1: Normalized performance metric of constrained cooperative navigation. Win rate (%) corresponds to the performance of the high-level policy, while $\min_{ij}\{d_{ij}\}$ corresponds to the performance of the low-level policy. The reported results are the mean over all the agents over five runs. First and second best results are represented with bold text.

Scenario		Ours			Non-hierarchical MARL			Hierarchical MARL		
		LPMARL	Dec LPMARL	No-LP	Qmix	MADDPG	MAAC	HSD	RODE	LDSA
$(M, N) = (3, 5)$	Win rate (%)	**85.8**	**75.2**	34.9	37.2	20.8	27.3	48.7	43.6	51.5
	$\min_{ij}\{d_{ij}\}$	0.12	0.24	0.42	0.30	0.43	0.23	0.19	0.29	0.17
$(M, N) = (5, 7)$	Win rate (%)	**82.5**	**71.2**	28.3	33.2	15.0	19.5	44.9	43.1	39.6
	$\min_{ij}\{d_{ij}\}$	0.11	0.34	0.53	0.29	0.48	0.22	0.18	0.38	0.28

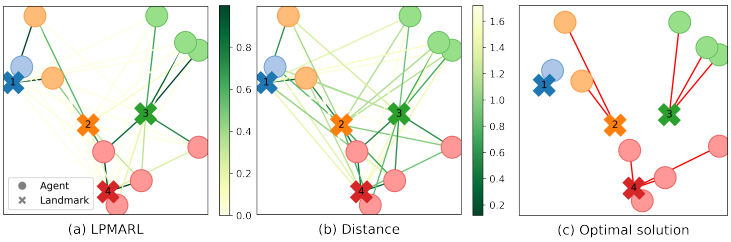

Figure 2: (a) Visualization of the learned coefficient of LPMARL. (b) Visualization of the agent-task distance matrix. (c) Optimal solution of LP with both input coefficients.

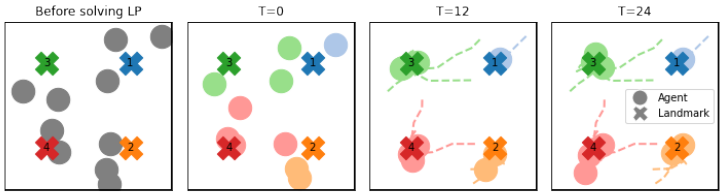

Figure 3: Visualization of transfer policy learned from $N = 5, M = 3$ to a task of $N = 10, M = 4$. The agent-landmark with the same color represents the result of the high-level assignment.

Table 1 shows performance of a constrained cooperative navigation when trained with $M, N = (3, 5), (5, 7)$. The results indicate that none of the baseline algorithms were able to effectively divide agents into groups without violating constraints, while the LPMARL was able to learn an optimal allocation and navigation policy, achieving 85.8% success rate in the training environment. Additional experimental results can be found in the Appendix F.1.

Note that the success of the agents in the constrained cooperative navigation task is not solely based on their proximity to the landmark, but also takes into account other factors. For example, even if the agents are able to navigate towards a landmark, they may not receive high rewards if that landmark is not the optimal one.

5.1.1 Interpretation of high-level policy

We compared LPMARL with LP-distance, which utilizes the distance matrix in solving the upper-level assignment problem, to interpret the learned optimization coefficient. Figure 2 visualizes the learned coefficient of LPMARL and the distance matrix for a specific problem instance. Comparing Figure 2 (a)-(b), we observe that the cost coefficient generating function $C_{\theta_c}(\cdot)$ generates coefficients resembling the distance matrix. While the learned coefficient function is sharper than the distance function, the optimal solutions of LPMARL and LP-distance are the same, as depicted in Figure 2 (c). This experiment showcases how LPMARL performs task assignments by considering the relationships between agents and tasks. Although this particular task could be solved straightforwardly using distance matrices, LPMARL demonstrates its capability to handle more complex relationships beyond distance. Further details on ablation studies concerning the high-level policy can be found in Appendix F.1.3.

5.1.2 Zero-shot policy transfer

To assess the size transferability of LPMARL, we conducted experiments in a constrained cooperative navigation environment with varying sizes. Due to the fixed input dimension of the baseline algorithm discussed in Section 5.1, the trained baseline model cannot be transferred to problems with a different number of agents. Therefore, we compared the performance exclusively with our ablation model.

The cost coefficient function and low-level policy, which are based on GNN, are designed to handle games with varying numbers of agents and tasks. To evaluate their transferability, we applied the policy learned from $3 \leq N, M \leq 10$ to a larger-scale problem. Figure 3 illustrates an example of the be-

Table 2: Success ratio (%) of the transferring policy learned on $N, M \in [3, 10]$. The gray and blue shaded cell represents the in-training distribution and out-of-training distribution respectively.

	LPMARL					Dec-LPMARL					No-LP				
$N \rightarrow$ $M \downarrow$	3	5	10	15	20	3	5	10	15	20	3	5	10	15	20
3	98.4	96.5	85.2	72.2	66.2	93.5	88.1	82.1	62.7	52.1	51.2	30.9	9.2	0.0	0.0
5	-	98.2	82.1	70.3	62.8	-	79.5	64.2	55.0	32.2	-	35.2	5.8	0.0	0.0
10	-	-	83.5	65.2	49.1	-	-	54.0	38.7	26.5	-	-	3.2	0.0	0.0
15	-	-	-	68.9	53.2	-	-	-	37.9	19.7	-	-	-	0.0	0.0
20	-	-	-	-	48.0	-	-	-	-	15.5	-	-	-	-	0.0

Table 3: Win ratio (%) on SMAC. The mean and standard deviation of the evaluation episodes are reported. First and second best results are represented with colored text.

Map name	LPMARL	Dec-LPMARL	No-LP	Qmix	SEAC*	RODE
3m (sparse)	**44.2**$\pm$3.7	**31.6**$\pm$2.0	8.4$\pm$1.3	10.4$\pm$3.0	8.3$\pm$2.8	18.6$\pm$2.1
2m_1z (sparse)	**44.3**$\pm$4.9	37.2$\pm$3.0	5.4$\pm$0.6	13.4$\pm$3.3	30.5$\pm$4.6	**35.1**$\pm$4.7
3s_5z (sparse)	**0.9**$\pm$0.5	0.0$\pm$0.0	0.0$\pm$0.0	0.0$\pm$0.0	0.0$\pm$0.0	**0.3**$\pm$0.0
8m (sparse)	**9.5** $\pm$0.5	**6.1**$\pm$0.5	0.0$\pm$0.0	3.1$\pm$0.3	-	2.6$\pm$0.2

* It is impossible to train SEAC on large problem instances due to the memory allocation problems.

havior of an LPMARL agent in a scenario with $N = 10$, $M = 4$, and $(k_1, k_2, k_3, k_4) = (1, 2, 3, 4)$. The landmark text indicates the capacity limit of each landmark. In the leftmost subfigure, representing the initial state of the environment, the circles represent agents, and the cross marks represent landmarks. In the second subfigure, the high-level policy assigns each agent to a landmark with the corresponding color. Subsequently, the agents navigate towards their assigned landmarks, leading to a successful state where all constraints are satisfied.

Table 2 presents the performance of the policy learned from $3 \leq N, M \leq 10$ when tested on $3 \leq N, M \leq 20$ without further training. LPMARL achieves a success rate of over 50% on out-of-training distributions, demonstrating its zero-shot transferability to larger scenarios.

5.2 StarCraft2 Micromanagement

The last environment we consider is the StarCraft Multi-Agent Challenge [Samvelyan et al., 2019]. In this environment, we can verify the dynamic goal assignment of the algorithms as the number of agents/enemies may change within the episode. The baselines we consider for SMAC are Qmix, RODE, and SEAC [Christianos et al., 2020].

Table 3 shows the final win ratio on SMAC tasks. In the sparse-reward setup, LPMARL stands out as the only method achieving meaningful win ratios across all scenarios. Particularly, in the *3s_5z* and *8m* environments, the win ratio of LPMARL surpasses other algorithms by at least threefold. Although it has a slightly lower win rate than its centralized version, Dec-LPMARL still has competitive performance compared to other baselines. More information including experimental results and video link can be found in Appendix F.2.

6 Conclusion and limitation

We proposed LPMARL, an LP-based hierarchical MARL approach, to effectively solve cooperative games with sparse rewards while optimally allocating agents to tasks. We demonstrated that LPMARL can decompose agents into sub-tasks across various environments.

We assume that the target problem is explicitly decomposed into multiple tasks. Even if this assumption does not hold, there is still potential for applying the proposed technique. For example, in prey–predator settings, it is possible to differentiate the prey's role into *chase* and *distract* and assign agents accordingly.

References

Sanjeevan Ahilan and Peter Dayan. Feudal multi-agent hierarchies for cooperative reinforcement learning. *arXiv preprint arXiv:1901.08492*, 2019.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *arXiv preprint arXiv:1909.01377*, 2019.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Nicolas Carion, Nicolas Usunier, Gabriel Synnaeve, and Alessandro Lazaric. A structured prediction approach for generalization in cooperative multi-agent reinforcement learning. *Advances in neural information processing systems*, 32:8130–8140, 2019.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. *arXiv preprint arXiv:2006.07169*, 2020.

Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Aaron Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. Mipaal: Mixed integer program as a layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1504–1511, 2020.

Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 2961–2970. PMLR, 2019.

Iou-Jen Liu, Unnat Jain, Raymond A Yeh, and Alexander Schwing. Cooperative exploration for multi-agent deep reinforcement learning. In *International Conference on Machine Learning*, pages 6826–6836. PMLR, 2021.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 2017.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304. PMLR, 2018.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Hongyao Tang, Jianye Hao, Tangjie Lv, Yingfeng Chen, Zongzhang Zhang, Hangtian Jia, Chunxu Ren, Yan Zheng, Zhaopeng Meng, Changjie Fan, et al. Hierarchical deep multiagent reinforcement learning with temporal abstraction. *arXiv preprint arXiv:1809.09332*, 2018.

Marin Vlastelica, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. Differentiation of blackbox combinatorial solvers. *arXiv preprint arXiv:1912.02175*, 2019.

Tonghan Wang, Heng Dong, Victor Lesser, and Chongjie Zhang. Roma: Multi-agent reinforcement learning with emergent roles. *arXiv preprint arXiv:2003.08039*, 2020a.

Tonghan Wang, Tarun Gupta, Anuj Mahajan, Bei Peng, Shimon Whiteson, and Chongjie Zhang. Rode: Learning roles to decompose multi-agent tasks. *arXiv preprint arXiv:2010.01523*, 2020b.

Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1658–1665, 2019.

Jiachen Yang, Igor Borovikov, and Hongyuan Zha. Hierarchical cooperative multi-agent reinforcement learning with skill discovery. *arXiv preprint arXiv:1912.03558*, 2019.

Mingyu Yang, Jian Zhao, Xunhan Hu, Wengang Zhou, Jiangcheng Zhu, and Houqiang Li. Ldsa: Learning dynamic subtask assignment in cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 35:1698–1710, 2022.

A Visualization of Extreme point of LP

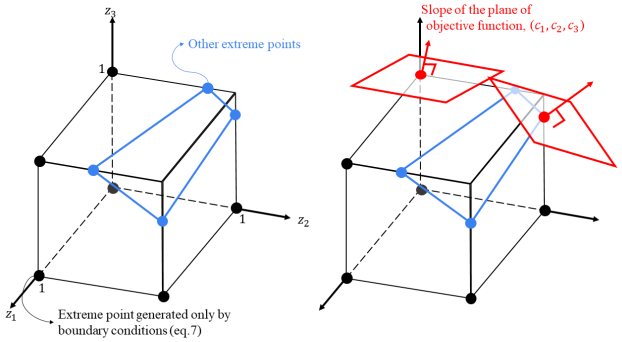

Figure 4: Visual description of extreme points

Figure 4 illustrates the extreme points of the linear programming in 3-dimensional case, where the decision variables are $(z_1, z_2, z_3) \in \mathbb{R}^3$. The boundary constraints are illustrated as the surface of the cube, and additional constraints other than boundary constraints are illustrated as the surface generated by blue lines. In Figure 4 (left), extreme points corresponding only to the boundary constraints are marked as the black dots, and the other extreme points are marked as the blue dots.

The objective of the optimization is to maximize $f(\boldsymbol{z}) = c_1 z_1 + c_2 z_2 + c_3 z_3$. The objective coefficients (c_1, c_2, c_3) determine the slope of the plane of the objective function, as illustrated by the red arrow on Figure 4. In Figure 4, we consider two possible optimization outcomes. For case (1), the optimal solution $z^* = (z_1^*, z_2^*, z_3^*)$ occurs only on the extreme points that are generated by the boundary points, and for case (2), z^* occurs on the other extreme points. In case (1), the optimal solution z^* is integer-valued. In our LP formulation, we have $N \times M$ boundary constraints and $N + M$ additional constraints. Thus, depending on the value of the objective coefficient and constraint coefficients, there may exist some integer-valued solution. In addition, the solution of the LP hanges discontinuously on the continuous change on the slope of the objective plane. For this reason, we need continuous interpolation of the gradient of the discontinuous surface $\frac{\partial \mathcal{J}_h}{\partial C}$.

B Policy gradient approximation for LP layer

The gradient of the high-level policy $\nabla_{\theta_h} \mathcal{J}_h(\theta_h)$ can be computed as the following chain rule:

$$\frac{\partial \mathcal{J}_h(\theta_h)}{\partial \theta_h} = \frac{\partial \mathcal{J}_h}{\partial Z^*} \cdot \frac{\partial Z^*(C)}{\partial C} \cdot \frac{\partial C}{\partial \theta_h} \tag{9}$$

We can compute the gradient $\frac{\partial Z^*(C)}{\partial \theta_h} = \frac{\partial Z^*(C)}{\partial C} \cdot \frac{\partial C}{\partial \theta_h}$ by differentiating the equality of the KKT conditions. Previous works Wilder et al. [2019], Ferber et al. [2020] proposed to add additional quadratic regularization term $\gamma||z||$ in the objective function (Eq. 4) to make LP as smooth QP, and induce non-singular Jacobian matrix to differentiate through the equality of the KKT conditions as in Amos and Kolter [2017].

However, although our high-level optimization problem is formulated as an LP, there exist extreme points generated only by the boundary constraints (Eq. 7). Thus, the optimal solution may be integer-valued on this extreme point as described in Appendix A. In this case, the optimal solution may not vary continuously with respect to the input; a small change can induce an abrupt change in the objective value. Thus, the $\frac{\partial Z^*(C)}{\partial C}$ be in the form of the piecewise constant, making it difficult to estimate the gradient. To solve this issue, Vlastelica et al. [2019] presented a piecewise linear surrogate loss surface that can approximate the original piecewise constant surface $\frac{\partial \mathcal{J}_h}{\partial C}$ of the combinatorial optimization layer. To obtain a meaningful differential value for this integer-solution, the gradient $\frac{\partial \mathcal{J}_h}{\partial C} = \frac{\partial \mathcal{J}_h}{\partial Z^*} \cdot \frac{\partial Z^*(C)}{\partial C}$ approximated by using the solution gap between the original solution of the optimization problem and the solution of the perturbed optimization problem, as:

$$\frac{\partial \mathcal{J}_h}{\partial C} \approx -\frac{1}{\lambda}(Z^* - Z_\lambda^*) \tag{10}$$

where Z^* and C are the optimal solution and coefficient used in the forward pass, respectively. Z^*_λ is the approximate solution computed as:

$$Z^*_\lambda = Z^*(C'_\lambda). \tag{11}$$

Here, C'_λ is the perturbed cost coefficient computed as

$$C'_\lambda := C + \lambda \cdot \frac{d\mathcal{J}_h}{dZ}(Z^*) \tag{12}$$

where λ is the hyperparemeter that scales the amount of perturbation while considering the gradient with respect to the solution Z^*. $\frac{d\mathcal{J}_h}{dZ}(Z^*)$ is the gradient of $\mathcal{J}_h$ with respect to solver output Z at given point Z^*. The amount of solution gap between the original solution and perturbed solution $Z^* - Z^*_\lambda$ is the slope of the piecewise linear function that will replace $\frac{\partial \mathcal{J}_h}{\partial C}$ evaluated at C. One can guarantee that the modified loss function is piecewise affine and similar to the original loss function (we refer to Vlastelica et al. [2019] for more detail).

C Detailed Training Derivations

C.1 High-level critic

The high-level critic $Q^h(\cdot;\phi_h)$ is trained to minimize the loss

$$\mathcal{L}(\phi_h) = \mathbb{E}\left[\left(Q^h(\boldsymbol{s}_t, a^h_{i,t};\phi_h) - y\right)^2\right], \tag{13}$$

$$y = r^h_t + \sum_i \gamma^\tau \max_{a^h_{i,t_\tau}} Q^h(\boldsymbol{s}_{t+\tau}, a^h_{i,t+\tau};\bar{\phi}_h), \tag{14}$$

where $\bar{\phi}_h$ is the target parameter. The temporal discounting accounts for high-level actions that last τ timesteps.

C.2 High-level policy gradient

The high-level policy gradient can be expressed as

$$\nabla_{\theta_h}\mathcal{J}_h(\theta_h) = \frac{\partial \mathcal{J}_h}{\partial Z^*} \cdot \frac{\partial Z^*(C)}{\partial C} \cdot \frac{\partial C}{\partial \theta_h}, \tag{15}$$

where Z^* is the LP solution given coefficient matrix C. Since $Z^*(C)$ is piecewise constant at integer extreme points, we follow Vlastelica et al. [2019] to approximate

$$\frac{\partial \mathcal{J}_h}{\partial C} \approx \frac{1}{\lambda}(Z^*_\lambda - Z^*), \tag{16}$$

with Z^*_λ being the solution of the perturbed LP.

C.3 Low-level critic

For agents grouped by high-level task j, the low-level critic Q^l is trained with

$$\mathcal{L}(\phi_l) = \mathbb{E}\left[\sum_{j\in\mathcal{M}}\Big(\sum_{i|a^h_i=j} Q^l(\boldsymbol{s}, a^l_i; a^h_i, \phi_l) - y\Big)^2\right], \tag{17}$$

$$y = r^l_j + \sum_{i|a^h_i=j} \gamma \max_{a^l_i} Q^l(\boldsymbol{s}', a^l_i; a^h_i, \bar{\phi}_l), \tag{18}$$

where r^l_j is the sub-group reward and $\bar{\phi}_l$ is the target parameter.

C.4 Low-level policy gradient

The low-level policy is trained via

$$\nabla_{\theta_l}\mathcal{J}_l(\theta_l) = \mathbb{E}\left[\sum_i Q^l(\boldsymbol{s}, a^l_i; a^h_i, \bar{\phi}_l) \cdot \nabla_{\theta_l} \log \pi^l(a^l_i|\boldsymbol{s}, a^h_i, \theta_l)\right]. \tag{19}$$

D Details on Dec-LPMARL

In Dec-LPMARL, the training process focuses on amortizing the high-level policy of LPMARL in a decentralized manner, eliminating the need for solving a centralized optimization problem during execution. To achieve this, Dec-LPMARL employs a decentralized high-level policy, denoted as $\hat{\pi}_i^h(a_i^h|o_i)$, which takes a local observation o_i as input and outputs the probability of choosing tasks (Figure 1). The difference between LPMARL and Dec-LPMARL lies in two aspects: (1) the construction of the graph, and (2) the training of the high-level policy.

First, in Dec-LPMARL, the graph used to generate cost coefficients is constructed in a different way. Dec-LPMARL constructs edge between entities within the observation scope, i.e., $\mathcal{G}_{dec} = (\mathcal{V}, \mathcal{E}_{dec})$, where $\mathcal{E}_{dec} \in \mathcal{E}$. This enables the agents to collectively construct the graph without relying on a centralized entity.

Second, the training of the high-level policy in Dec-LPMARL involves amortizing the pre-trained LPMARL high-level policy. This is done using a behavior cloning loss, which aims to minimize the negative log likelihood of the pre-trained LPMARL policy's task selection given the local observation. Specifically, the behavior cloning loss can be formulated as follows:

$$\mathcal{L}_{\text{BC}} = -\mathbb{E}_{\tau\sim D,i,t}\left[\log \hat{\pi}_i^h(a_i^{h*}|o_i)\right] \tag{20}$$

where $\hat{\pi}_i^h(a_i^{h*}|o_i)$ represents the predicted probability of choosing the expert (LPMARL) action a_i^{h*} at time step t given the agent's local observation o_i using $\hat{\pi}_i^h$. By training the decentralized high-level policy in this way, Dec-LPMARL can effectively leverage the knowledge encoded in the LPMARL policy while operating in a decentralized manner.

E Hyperaparemeters and Experimental details

The hyperparameters of LPMARL used in the experiment are summarized in Table 4. We used an CVXPY solver Diamond and Boyd [2016] to solve the linear programming in Section 3.

Table 4: Hyperparemeters of LPMARL

Hyperparemter	Values
MLP units for GNN, $f(\cdot;\theta_g^{\mathcal{N}\mathcal{M}})$, $f(\cdot;\theta_g^{\mathcal{M}\mathcal{M}})$, $f(\cdot;\theta_g^{V})$	[32,32]
MLP units for coefficient matrix, $f(\cdot;\theta_c)$	[64,64]
MLP units for policy network, $f(\cdot;\theta_h)$, $f(\cdot;\theta_l)$	[64,64]
MLP units for critic network $f(\cdot;\phi_h)$, $f(\cdot;\phi_l)$	[64,64]
Nonlinear activation	LeakyReLU, negative slope=0.01
learning rate	10^{-3}
Discount rate, γ	0.99
λ	20
Optimizer	Adam

For other baselines algorithms, we used a two-layer feed-forward fully-connected network with a 64-dimensional hidden layer and ReLU activation. Batches of 32 episodes are sampled from the replay buffer. The optimizer, learning rate, and discounting rate γ of the other algorithms are set to be the same as in Table 4. Experiments are carried out on NVIDIA RTX A6000.

In the experimental environment, the rewards are provided differently to the algorithms. For the hierarchical MARL algorithms and LPMARL, each level of reward described above is used to train the corresponding policy level. For the non-hierarchical MARL algorithms, a weighted sum of the high- and low-level rewards is used to train the policy.

E.1 Cooperative navigation

We used 50,000 episodes where the maximum timestep of the each episode is 50. For every algorithm, we set the size of the replay buffer as 5,000 with a batch size of 32 to train. For LPMARL, we set k_i, the capacity constraint coefficient of Eq. 5, as 1 at every scenario. The initial location

of the agents and landmarks are randomly spawn on $[-1, 1] \times [-1, 1]$ and $[-0.8, 0.8] \times [-0.8, 0.8]$, respectively. The reward conditions of the experiment is specified in Table 5.

Reward setting		Hierarchical MARL, LPMARL	Non-hierarchical MARL
Dense setting	High-level reward	+1 when all agents reaches landmark	$0.5\times$High-level reward $+0.5\times$Low-level reward
	Low-level reward	+1 when each agent reaches landmark distance and collision-related reward	
Sparse setting	High-level reward	+1 when all agents reaches landmark	$0.5\times$High-level reward $+0.5\times$Low-level reward
	Low-level reward	+1 when each agent reaches landmark	

Table 5: Reward setup of cooperative navigation environment.

E.2 Constrained Cooperative navigation

We used 50,000 episodes where the maximum timestep of the each episode is 50. For LPMARL, we set k_i, the capacity constraint coefficient of Eq. 5, to be the same as the landmark capacity. The initial location of the agents and landmarks are randomly spawn on $[-1, 1]\times[-1, 1]$ and $[-0.8, 0.8]\times[-0.8, 0.8]$, respectively.

We set the capacity of the landmarks as an integer partition of N into M groups, i.e., $\sum_{j=1}^{M} k_j = N$ where k_j is the number of agents that can be accommodated by landmark j. The capacity of the landmark is randomly determined at the beginning of each episode. Agents can observe the capacity and the position of each landmark within the observation range.

E.3 StarCraft Multi-Agent Challenge

We train all models over 100,000 episodes. For every algorithm, we set the size of the replay buffer as 5,000 with a batch size of 100 to train. For LPMARL, we set k_i, the capacity constraint coefficient of Eq. 5, as $\lceil \frac{\mathcal{M}}{2} \rceil$ at every scenario.

We consider the following SMAC scenarios:

- *3m*: 3 marines ($N = 3$) versus 3 marines ($M = 3$). The episode limit is 60 timesteps.
- *2m_1z*: 2 marines ($N = 2$) versus 1 zealot ($M = 1$). The episode limit is 150 timesteps.
- *3s_5z*: 3 stalker ($N = 3$) versus 5 zealots ($M = 5$). The episode limit is 150 timesteps.
- *8m*: 8 marines ($N = 8$) versus 8 marines ($M = 8$). The episode limit is 150 timesteps.

Difficulty levels of the scenarios are all set to be *harder (6)*.

F Additional Experiment Results

F.1 Cooperative Navigation

F.1.1 Training curve

Training curve on constrained cooperative navigation is shown in Figure 5. Figure 5 shows the high- and low-level reward curve over the episodes when the algorithm is trained when N and $M = 3$.

F.1.2 Amortization accuracy

Figure 6 illustrates the performance of the Dec-LPMARL algorithm in terms of prediction accuracy across various training episodes. The results demonstrate that the algorithm demonstrates high performance, with prediction accuracy exceeding 90%, when dealing with simple tasks such as dense-3. However, as the complexity of the optimization problem increases, the prediction accuracy of the Dec-LPMARL algorithm decreases to 75% (3M_5N). This outcome highlights the strong correlation between the performance of the Dec-LPMARL algorithm and the performance of its amortized LP layer. The performance of the Dec-LPMARL algorithm can potentially be improved by enhancing its ability to learn the optimal solution of constrained optimization.

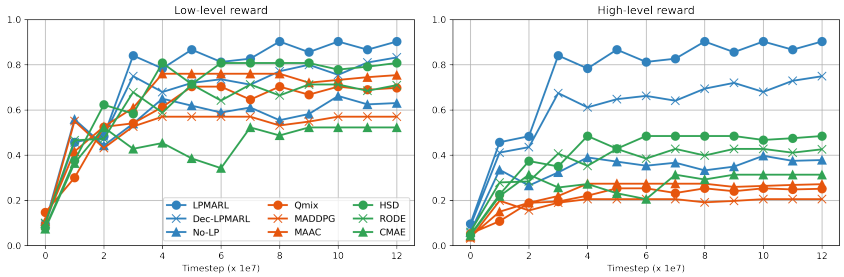

Figure 5: Low-level (left) and high-level (right) reward curve. The reported reward is the mean over all the agents over the timestep per episode.

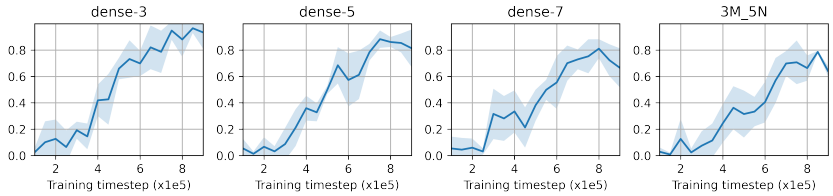

Figure 6: Fitting accuracy of Dec-LPMARL

F.1.3 Effectiveness of LP layer

We consider the following ablations of high-level policy to examine the effectiveness of using LP as a high-level policy. **LP-distance** assigns an agent to a goal using the distance matrix as the objective coefficient of LP. Therefore, agents are assigned to landmarks that minimize the sum of the distance of all the agents. **Greedy** is a greedy assignment, where each agent chooses the closest landmark individually. The network structure for low-level policies is identical for all the high-level ablations.

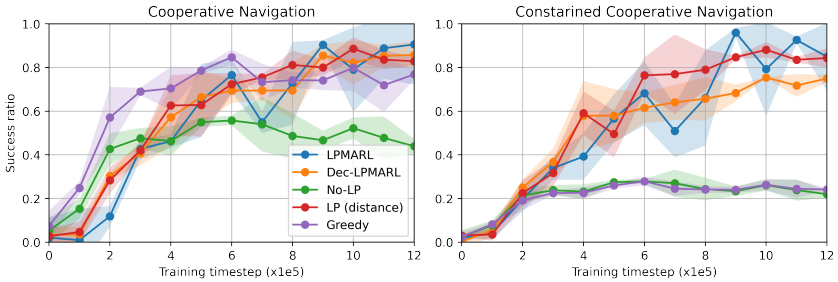

Figure 7: Training curve of ablations of high-level policy on cooperative navigation (left) and constrained cooperative navigation (right)

Figure 7 shows the results of ablation studies. LP-distance uses hand-designed features to induce good coordination for tasks where proximity plays essential roles, such as cooperative navigation. However, devising a hand-defined rule for a complex task is challenging. In such cases, the proposed algorithm that constructs the cost matrix considering the global state can play an important role in deriving an effective policy.

F.1.4 Effect of sparse reward help for training non-hierarchical MARL

In cooperative navigation, although non-hierarchical MARL can be trained only with dense reward (distance and collision-related reward), we use a weighted sum of the high-level (sparse) and low-level reward to compare the performance fairly. If only dense reward is used for non-hierarchical MARL, their performance degrades. Table 6 compares the success ratio of using only dense reward for non-hierarchical MARL algorithms on cooperative navigation environment.

	Qmix		MADDPG		MAAC	
	T+E	E	T+E	E	T+E	E
Dense-3	82.4	87.0	85.7	82.1	89.1	85.5
Dense-5	62.0	60.9	21.3	32.8	74.5	76.2
Dense-7	49.1	43.0	5.7	12.2	65.7	52.0

Table 6: Success ratio when cooperative MARL algorithms are only trained with environment reward. (T+E) indicates (T)ask-dependent reward + (E)nvironmental reward, and (E) indicates environmental reward.

F.2 SMAC environment

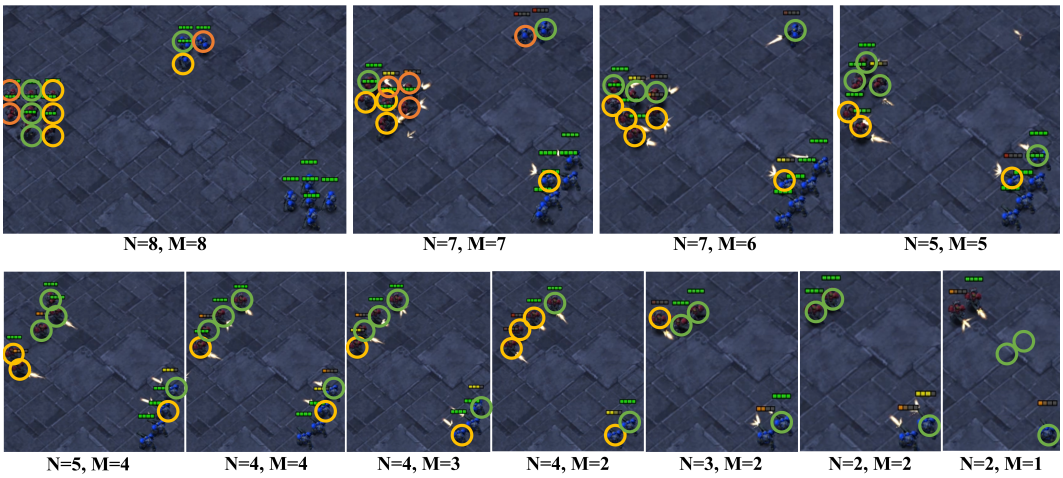

Figure 8: Visualization of high-level assignment on StarCraft environment (*8m*).

Figure 8 visualizes the high-level assignment of LPMARL on *8m* environment. We divided the enemy units into two groups, with three and five units, to clearly see how LPMARL allocates agents to enemies. Each sub-figure of Figure 8 shows the high-level assignment result on timestep when the event (when M or N changes) occurs. The agent-task with the same color represents the result of the high-level assignment.

In the figure, we can observe that the LPMARL agent sequentially kills all the enemies by focusing fire from the nearest enemy (task). Also, LPMARL does not always assign only the closest enemy to the ally but assigns agents to the enemy with the lowest health level ($N = 5, M = 4$). Further visualization videos can be found at the following link.

G Codes

The code can be found at the following link.